METHOD OF DYNAMIC ADAPTATION FOR JITTER BUFFERING IN PACKET NETWORKS

Field of the Invention

The present invention relates generally to digital transmission systems, and more particularly to a method for reducing perceived network jitter by varying the amount of delay induced by a jitter buffer in a packet network.

Background of the Invention

In packet-switched voice communication, maximum voice quality is achieved when voice packets arrive in the order that they were transmitted, at the exact rate that they are transmitted, and with the shortest possible transmission delay. However, the nature of data transmission in packet-switched or IP networks inherently gives rise to transmission delays (i.e. jitter) that may vary widely due to available bandwidth, number of nodes traversed by the packets, network congestion, etc. Packets can also be duplicated by network switching equipment or dropped. In addition, as a consequence of packet routing policies that are commonly employed, it is also possible for packets to arrive at an endpoint in a different order than they were transmitted.

It is known in the art to use data buffers when receiving voice packets, to ameliorate the effects of network jitter. For example, US Patent No. 6,603,749, entitled Adaptive Buffer Management for Voice Over Packet network, by Andre Moskal and Andre Diorio, discloses a fixed length adaptive buffer, wherein the length of the buffer is a compromise between introduced delay and induced packet loss due to underflow or overflow. The fixed length adaptive buffer is effective on well-conditioned networks. However, on less well managed networks (e.g. jitter greater than 40 ms) such as the Internet, the jitter often exceeds the size of the buffer. Consequently, the buffer actually introduces additional packet loss.

Accordingly, much research has centered on methods that dynamically adapt the buffer length according to current network conditions.

Dynamic jitter buffers reduce the effects of variable transmission delay by introducing additional delay at the packet receiver. This has the benefit of requiring no special consideration at the packet transmitter. Dynamic jitter buffers reduce perceived network jitter by varying the amount of delay induced by the jitter buffer according to detected changes in network transmission delay.

Several approaches in the literature use adaptation to dynamically adjust the buffer to current changes in network delay. Adaptation techniques include LMS, neural networks, and fuzzy logic. Other methods use state machines, as set forth in Dynamic Jitter Buffering for Voice-Over-IP and Other Packet-Based Communication Systems (US Patent Application No. 2003026275). Unfortunately, these tend to suffer from implementation complexities.

Summary of the Invention

According to the present invention, a simple, state-less adaptation control algorithm is provided, with a fast attack and a slow decay time to track delay changes in the network. The principal function of the algorithm for controlling the jitter buffer is to minimize the delay within the buffer (at the expense of occasional buffer underflow). Traditionally, jitter buffers attempt to smooth out jitter by preventing underflow, whereas minimizing the delay within the buffer is a secondary consideration. Although such prior art buffers prevent more underflow errors than the system of the present invention, they tend to introduce longer delays. For example, the fixed buffer discussed above in connection with U.S. Patent Application No. 6,603,749, is an example of a buffer control algorithm that maintains an average midpoint within the buffer.

Brief Description of the Drawings

An embodiment of the present invention will now be described, by way of example only, with reference to the attached Figures, wherein:

Figure 1 is a schematic representation of an adaptive jitter buffer according to the present invention;

Figures 2A, 2B and 2C are flowcharts showing steps for enqueueing and dequeueing data into and out of the jitter buffer of Figure 1, including adjustment of watermark levels governing the rate of dequeueing data based on network conditions, according to the present invention; and

Figure 3 is a diagram showing an example of the buffer contents over time, including the adjustment of watermark levels, according to the present invention.

Detailed Description of the Invention

Figure 1 shows an adaptive jitter buffer according to the present invention, for enqueueing data packets that have been subjected to unsmoothed jitter and dequeueing the data at a steady rate so that the dequeued data is subjected to smoothed jitter. The total number of packets capable of being stored in the buffer is represented by the maximum buffer size, while the minimum number of packets is LWM_TH. The "count" gives the number of packets in the buffer at any time, while the low water mark (LWM) indicates how far the buffer is away from underflow, and the high water mark (HWM) tracks the delay within the buffer.

Turning to Figure 2, upon receipt of a data packet an Enqueue event is initiated (step 201 of Figure 2A). The received packet is loaded into the buffer in correct sequence number position, and "count" is incremented to reflect the current number of packets (step 203).

If count exceeds HWM then HWM is set to count (step 205). If the count does not exceed HWM, or after step 205, a determination is made as to whether count is less than LWM (step 209). If it is, then LWM is set to count (step 211). If count is not less than LWM, or after step 211, the average count is computed for statistics purposes (step 213).

Optionally, a dequeue event (Figure 2B) is generated (step 215), and the Enqueue process ends (step 217).

The Enqueue event may be implemented in software as follows:

```
 Enqueue
For every packet
   Enqueue packet in correct seq num position
   Increment count   // number of packets in queue

   /* Adjust watermarks, if required (fast attack) */
   IF count > HWM
      HWM = count    // adjust HWM
   ENDIF

   IF count < LWM
      LWM = count
   ENDIF

   Average current count in buffer
ENDFOR
```

A Dequeue event (Figure 2B) may be generated after each Enqueue (step 215), and is generated in any event each Dequeue Time Tick (e.g. 20 ms).

First, the current Dequeue TimeStamp (msCurrent) is obtained and the difference (msDiff) between the current and previous TimeStamps is calculated, representing the time since the last Dequeue event (step 221).

If msDiff is greater than the Dequeue Time Tick, a determination is made as to whether the buffer is empty (step 225). If not, a DequeueBuffer event is generated (Figure 2C), and both msDiff and Dequeue Time Stamp are updated.

If the buffer is empty (indicating an underflow condition), data is inserted into the packet stream by invoking well known packet loss concealment, an underflow counter is incremented and the Dequeue Time Stamp is updated (step 229).

If msDiff is not greater than the Dequeue Time Tick, or after step 229, a determination is made as to whether a predetermined time period (e.g. 2s) has elapsed since the last slip adjust (step 231).

If yes, then if LWM exceeds LWM_TH (step 233), then a DequeueBuffer event is generated and a shrink counter is incremented (step 235). Next, or if LWM does not exceed LWM_TH, then the watermarks are re-initialized to the count value and a Shrink timestamp is updated (step 237).

If the time to slip adjust has not yet elapsed (step 231), or in any event after step 237, a determination is made (step 239) as to whether the buffer is overflowing (i.e. count>max buffer size). If yes, a DequeueBuffer event is generated and an Overflow Counter is incremented (step 241).

If the buffer is not overflowing, or in any event after step 241, the Dequeue event ends (step 243).

The Dequeue event may be implemented in software as follows:

```
Dequeue

      FOR every enqueue or 20ms tick  // JB dequeue triggered
         Get current time stamp
         Calculate time difference (msDiff) since last dequeue
         WHILE msDiff > 20ms
            DequeueBuffer
            IF queue empty,
               Save current dequeue time
               Increment underflow counter
               break out
            END
            msDiff = msDiff - 20ms
            Update current dequeue time
         END WHILE

         // Check whether we have to slip adjust or are still in
      overflow
         FOR every jitter_Q_shrink_rate (currently 2s) time event
            IF LWM > LWM_TH (currently 1) (slow drain, deacy)
               DequeueBuffer  // dequeue another packet, slip adjust
                  Increment shrink counter
            ENDIF

            /* Re-initialize watermarks */
            LWM = count
            HWM = count

            Update shrink Timestamp
         ENDFOR

         IF num of packets > Jitter size   // Overflow checked on
      every enqueue and dequeue tick
            DequeueBuffer // Dequeue another packet
            Increment overflow counter
         ENDIF
      ENDFOR
```

The DequeueBuffer event (Figure 2C) governs the unloading of packets from the buffer. Following instantiation of the DequeueBuffer event

(step 245), the data packets or dequeued from the buffer and count is decremented accordingly (step 247).

If count exceeds HWM, then HWM is set to count (step 253). If not, and in any event after step 253, a determination is made (step 257) as to whether count is less than LWM. If yes, then LWM is set to count. If count is not less than LWM, and in any event after step 253, then the DequeueBuffer event ends (step 261).

The Dequeue event may be implemented in software as follows:

```
DequeueBuffer
BEGIN
   Dequeue data
   Decrement count    // number of packets in queue


   /* Adjust watermarks, if required (fast attack) */
   IF count > HWM
      HWM = count     // adjust HWM
   ENDIF

   IF count < LWM
      LWM = count
   ENDIF
END
```

From the foregoing description, it is apparent that the algorithm for controlling enqueueing and dequeueing of data packets according to the present invention minimizes the delay within the buffer by using a quick 'attack', at the expense of preventing buffer underflow. Since it is very difficult to predict network behavior, control of the buffer is biased towards introducing a minimum delay by adapting quickly to large jitter events and by inserting additional packets using packet loss concealment during buffer underflow. After a large jitter event, the buffer contains several packets and the delay introduced by the buffer is equivalent to the jitter length. When network conditions normalize, the delay within the buffer is far larger than is required (i.e. the actual current jitter is smaller than the buffer delay). The buffer is drained slowly (at the slip or drain rate, currently set at 2s). Consequently, control of the buffer is characterized by a slow decay time.

Figure 3 shows typical behaviour of the buffer under the enqueue and dequeue control algorithms of Figure 2, including adjustment of watermark

levels over time. It will be noted that the buffer control is dependent only on the low water mark (LWM), and that data is inserted when the buffer is in underflow (count = 0 and LWM = 0). The draining of data occurs at the drain rate when count exceeds LWM. The high water mark (HWM) is tracked for statistics purposes only, and is not used to control the buffer.

It will also be noted from Figure 3 that after a network congestion event, which forces the buffer into underflow (just prior to time stamp TS1), the buffer accepts all data after the congestion disappears (fast attack following TS1). Then, slowly over time, the delay is again drained out of the buffer (TS2, TS3, TS4). Normal operation (no attack, no drain) is when the LWM returns to below LWM_TH.

It will be appreciated that, although embodiments of the invention have been described and illustrated in detail, various modifications and changes may be made.

In one alternative embodiment, the number of packets (count) is averaged within the Enqueue event and the watermarks are adjusted relative to this average value. Specifically, the LWM is adjusted upwardly if the average count exceeds LWM, and the high water mark is adjusted downwardly if the average count is less than HWM. In other words, the watermarks decay towards the average count in the buffer.

According to a second alternative embodiment, the decay adjustment may be performed during the Dequeue event, on every slip adjust event, in which case no average count is calculated.

Also, different drain strategies can be used than as set forth herein. For example, a faster drain rate may be used when the delay within the buffer is long and a slower rate used when the delay is short. Also, since the high water mark is an indication of the delay within the buffer, it can be used as well in controlling the buffer.

Different implementations may be made by those familiar with the art, without departing from the scope of the invention as set forth in the claims appended hereto.